Efficient Light Propagation for Multiple Anisotropic Volume Scattering

Nelson Max

University of California, Davis, and

Lawrence Livermore National Laboratory

Abstract

Realistic rendering of participating media like clouds requires multiple anisotropic light scattering. This paper presents a propagation approximation for light scattered into M direction bins, which reduces the “ray effect” problem in the traditional “discrete ordinates” method. For a regular grid volume of n^3 elements, it takes $O(M\,n^3 \log n + M^2\,n^3)$ time and $O(M\,n^3 + M^2)$ space.

This document is reprinted from the proceedings of the Fifth Eurographics Workshop on Rendering, Darmstadt, Germany, June 13 - 15, 1994

1. Introduction

To render realistic images of clouds, one must take into account absorption and multiple scattering of incoming illumination. In addition, to produce the bright edges surrounding a cloud when the sun is behind it, one must account for the anisotropic, mainly forward, scattering of light from the water droplets.

In 1984, Jim Kajiya and Brian Von Herzen [Kaj84] proposed two methods for rendering clouds. The first was the two-pass “slab” method, which accounted only for single scattering. The first pass deposited flux from the light source into the cloud voxels one horizontal layer at a time, taking into account the attenuation by the opacity in each layer. The second pass gathered the scattered flux along each viewing ray, taking into account the attenuation between the scattering event and the viewpoint. Voss [Voss83] used a similar method to produce fractal clouds in terrain scenes. Nishita, Miyakawa, and Nakamae [Nish87] have considered anisotropic single scattering in fog, and Inakage [Inak89] has included cases where the density and phase function of the scattering material varies from point to point. Kaneda *et al.* [Kan90] also simulate anisotropic scattering in clouds and fog, including one case of double scattering: first Raleigh and Mie scattering to determine a fixed sky illumination, and then one more scattering of this illumination within a fog. Related work was also done by [Blas93].

Kajiya's second method was an application of the multiple scattering ideas of Chandrasekhar [Cha50], which use spherical harmonics to expand, at each point, the light intensity as a function of direction. The scattering phase function is also expanded in spherical harmonics, resulting in a set of coupled partial differential equations for the spherical harmonic coefficients of intensity as functions of the spatial coordinates. Kajiya attempted to solve these equations for the case of isotropic scattering, but it is unclear whether he succeeded, since all the pictures in [Kaj84] were produced by the simpler "slab" method.

The transport equations Kajiya used, described in the next section, have a long history in radiation heat transfer in mechanical engineering, and in particle transport in nuclear engineering. Siegel and Howell [Sie92] give a good summary of solution techniques. Holly Rushmeier [Rush87, Rush88] applied two of these solution techniques to computer rendering of *participating* (*i.e*. absorbing, emitting, and scattering) media. One was the Monte Carlo method, where a random collection of photons or flux packets are traced through the volume, undergoing random scattering and absorption. This method can accurately model all the physics of scattering, but may take an impractical number of random trials to converge to a useful solution.

The other was the *zonal* method for isotropic scattering only, which divides the volume into a number of *finite elements* which are assumed to have constant radiosity. This requires the calculation of a *form factor* between every pair of elements. In a cube of $N = n^3$ elements, there will be $N^2 = n^6$ such pairs of elements. In the *Galerkin* finite element scheme, each form factor involves a double integral over points in both elements, as well as along the path between the two points, giving a total of 7 integration variables. Rushmeier approximates this by an inverse square factor and a 1-D integral of opacity along the path connecting the element centers. If each of the $O(n)$ intervening elements has different scattering properties, this 1-D integral takes time $O(n)$. Using an iterative method for solving the resulting matrix equation which converges in $O(1)$ iterations, the total computational cost is $O(n^7)$. This cost can be reduced somewhat by grouping adjacent elements into larger interaction pairs, in the style of Hanrahan, Salzman, and Aupperle [Hanr91], as was done by Bhata [Bhat93]. Rushmeier [Rush88] also considers anisotropic scattering, but only in the single scattering case.

Zhiquiang Tan [Tan89] applied the ideas of finite element analysis to the solution for the spherical harmonic coefficients in the case of multiple anisotropic scattering. If there are M terms in the expansion, this results in a matrix of size M^2N^2. Tan uses the *point allocation* (or *point collocation*) method, which allows the representation of non-constant basis functions. He points out that this requires integrals over only one 3-D position, reducing the number of integration variables by 3. This simplification has been misinterpreted by Siegel and Howell [Sie92], who incorrectly claimed that the method is $O(N)$. Bhata [Bhat92] has applied this method to computer rendering, but could deal with only a small number of voxels, due to the $O(n^7 + M^2n^6)$ cost.

Another approach is to allocate the radiosity leaving each volume element into a collection of M direction bins of constant intensity. Assuming the interaction between two elements involves only one direction bin for flux transit (reasonable only for distant pairs of elements), this reduces the number of non-zero matrix elements to MN^2, and the cost to compute them to $O(n^7 + Mn^6)$.

Sparse matrix solution methods are then available, as in Immel *et al*. [Imm86].

In the *discrete ordinates* method in radiation transfer, [Sie92, Chan50], the M direction bins are represented by M discrete directions, chosen to give optimal Gaussian quadrature in the integrals over a solid angle. Lathrop [Lath68] points out that this process produces ray effects, because it is equivalent to shooting the energy from an element in narrow beams along the discrete directions, missing the regions between them. He presents modifications to avoid these ray effects, but the resulting equations are mathematically equivalent to the ones mentioned above for the spherical harmonic coefficients. This implies that M properly distributed direction bins specify the directional intensity distribution to the same detail as M spherical harmonic coefficients.

The current paper presents an approximation to the discrete ordinates method, which reduces the ray effect by shooting radiosity into the whole solid angle bin, instead of in a discrete representative direction. As a shooting method, it is similar to the progressive radiosity method of Cohen *et al.* [Coh87], and can be shown to converge for albedo less than one. (See [Gort93] and section 6 below.) Patmore [Patm93] has used a discrete ordinates shooting algorithm (subject to ray effects) for a multiple-scattering rendering of clouds, and his paper inspired the current one. Langer *et al*. [Lang93] have implemented the discrete ordinates method on a massively parallel SIMD machine, and included surface reflections. (See section 10 below.)

My chief enhancement is to spread the shot radiosity throughout the direction bin in an efficient way which handles a whole plane of source elements simultaneously, while reducing the ray effect. Another enhancement treats multiple scattering within a single receiving element before the next shooting step. I use $O(MN)$ space to store the total radiosity in each direction bin at each element, and also the unshot radiosity. The direction-bin-indexed matrix representing the anisotropic scattering function takes an additional $O(M^2)$ space. The computation for each pass through the M shooting directions takes time $O(Mn^3 \log n + M^2 n^3) = O(MN \log N + M^2 N)$. This large speedup compared to the other methods discussed can only be achieved with a regular cubical grid. Since it makes essential use of the homogeneity of the grid, my method will not work on more general finite element meshes.

2. Transport Equations

In thermal radiation heat transport, a participating medium which absorbs radiation heats up, and re-emits "black body" radiation isotropically. This effect is usually not important in rendering, and I will neglect it below for simplicity, and deal only with absorption and scattering. More complete discussions are available from [Sie92] and [Rush88].

Let $I(x,\omega)$ be the intensity at position x in direction ω, and let $k_t(x)$ extinction coefficient of the participating medium. This is the total opacity (absorption plus scattering) per unit length so $k_t(x)\ I(x,\omega)\ ds$ is the intensity removed along an infinitesimal ray segment ds at x. Let the *albedo*, a, be the fraction of this removed intensity scattered in other directions, and let the *phase function*, $f(\omega, \omega')$, be the directional distribution function for this scattered intensity, so that $\int_B f(\omega, \omega')\, d\omega$ is the fraction of the scattered intensity from direction ω' that ends up in solid angle B. Then

$$ak_t(x)\,ds \int_{4\pi} f(\omega', \omega)\, I(x, \omega')\, d\omega'$$

is the intensity scattered into the direction ω along the ray segment *ds* from other directions ω′ in the 4π unit sphere. (This is the *source function* [Sie92] in the absence of volume emission.) The integro-differential equation for *I*(*x*,ω) is thus

$$\frac{dI(x,\omega)}{ds} = -k_t(x)\, I(x,\omega) + ak_t(x) \int_{4\pi} f(\omega, \omega')\, I(x, \omega')\, d\omega'.$$

Using an integrating factor (see [Sie92], [Rush88], or [Will92]), this can be integrated along a path $x'(s) = x - s\,\omega$, from $x = x'(0)$ to $x_0 = x'(s_0)$ at the edge of the medium, to give the integral form

$$I(x,\omega) = I(x_0,\omega) \exp\left(-\int_0^{s_0} k_t(x'(s))\, ds\right)$$

$$+ a\int_0^{s_0} \left(k_t(x'(s)) \exp\left(-\int_0^{s} k_t(x'(t))\, dt\right) \int_{4\pi} f(\omega', \omega)\, I(x'(s), \omega')\, d\omega'\right) ds. \qquad (1)$$

Now assume that the region under study is divided into a collection of cubical volume elements V_k, which I also call cells, voxels, or, in 2-D, pixels. Assume that the unit sphere is divided into a number of direction bins B_l, and that *I*(*x*,ω) is constant for *x* in V_k and ω in B_l. In the implementation, these constant values are represented by a matrix `through[k][l]`, which stores the intensity multiplied by the solid angle of bin B_l, `size[l]`. I assume that the extinction coefficient k_t is constant in each element V_k, and stored in an array `kt[k]`. The input values for `kt[k]` are produced by a cloud modeler, described briefly later. I assume that the albedo *a* is constant everywhere, to avoid creating an extra array. Let *x* lie in cell V_i, and ω lie in angle bin B_j. Then with these assumptions, we can integrate equation (1) over B_j to get

$$\texttt{through[i][j]} = \int_{B_j} I(x,\omega)\, d\omega \qquad (2)$$

$$= \int_{B_j} d\omega \left\{ \frac{\texttt{through[}n(s_0)\texttt{][}l(\omega)\texttt{]}}{\texttt{size[}l(\omega)\texttt{]}} \exp\left(-\int_0^{s_0} \texttt{kt[}n(s)\texttt{]}\, ds\right)\right.$$

$$\left. + a\int_0^{s_0} \left[(\texttt{kt[}n(s)\texttt{]}) \exp\left(-\int_0^{s} (\texttt{kt[}n(t)\texttt{]})\, dt\right) \int_{4\pi} f(\omega, \omega') \frac{\texttt{through[}n(s)\texttt{][}l(\omega')\texttt{]}}{\texttt{size[}l(\omega')\texttt{]}} d\omega' \right] ds \right\}$$

where *n*(*s*) is the index of the volume element containing *x*′(*s*), and *l*(ω) is the index of the angle bin containing ω. Suppose, for simplicity, that all rays from *x* to cell V_l lie in angle bin *j*. (The algorithm described in the following sections takes special account of interactions involving two or more bins.) Geometric arguments (see [Rush88]) show that

$$Geom_{ijk} = \int_{B_j} d\omega \int_{n(s)=k} ds \cong \frac{\text{Volume } (V_k)}{r^2} \qquad (3)$$

where r is the distance between the centers of cells j and k. Thus the multiplier giving the contribution of `through[k][l]` to the last term in equation (2) is the “form factor”

$$F_{klij} = a\frac{\text{Volume }(V_k)}{r^2}\texttt{kt[k]}\exp\left(-\int_0^r(\texttt{kt[}n(t)\texttt{]})\,dt\right)\int_{B_l}d\omega'\frac{f(\omega,\omega')}{\texttt{size[l]}}\,. \tag{4}$$

I precalculated by Simpson’s rule integration the $M \times M$ matrix version of the phase function:

$$\texttt{bintobin[l][j]} = \int_{B_j}d\omega\int_{B_l}d\omega'\frac{f(\omega,\omega')}{\texttt{size[l]}}$$

giving the fraction of the flux from bin `l` directions which scatters into bin `j`. Replacing the factor $\int_{B_m}d\omega'\frac{f(\omega,\omega')}{\texttt{size[l]}}$ in equation (4) by its average value $\frac{\texttt{bintobin[l][j]}}{\texttt{size[j]}}$, we get

$$F_{klij} \cong a\frac{\text{Volume }(V_k)}{r^2\texttt{size[j]}}\texttt{kt[k]}\exp\left(-\int_0^r(\texttt{kt[}n(t)\texttt{]})\,dt\right)\texttt{bintobin[l][j]}\,. \tag{5}$$

In the implementation, I take the unit of length to be the side of a cubic cell V_k, so that the factor Volume(V_k) drops out. Note that this effects the extinction coefficients `kt[n]`, whose units are inverse length.

Using these form factors, one can write the usual system of linear equations for the unknown fluxes `through[k][l]`. I have developed an approximate solution method which accumulates opacity on the fly, as the flux is propagated in a shooting procedure. As in progressive radiosity for surface illumination [Coh88], I use an auxiliary array `unshot[k][l]` of size $MN = Mn^3$, to store the flux waiting to be propagated, and need not store the $M^2N^2 = M^2n^6$ form factors.The difficult part in evaluating equation (5) is in computing $\exp\left(-\int_0^r(\texttt{kt[}n(t)\texttt{]})\,dt\right)$ by integrating along a straight line joining the pixel centers. My method approximates each such term as a weighted sum of similar terms, obtained by integrating over piecewise linear paths that lie near the straight line. (See figure 4.) This permits sharing of calculations to compute the effect of the M^2N^2 form factors in time O($MN \log N + M^2N$).

For my test images, I used the Henyey-Greenstein phase function [Heny40]

$$f(\omega,\omega') = \frac{1}{4\pi}\frac{1-g^2}{(1+g^2-2gx)^{3/2}}$$

where x is the dot product of the two unit direction vectors ω and ω′, and g is an adjustable parameter between -1 and 1, which is positive for forward scattering, negative for backwards scattering, and 0 for isotropic scattering. For an appropriate choice of g, this is a good approximation to the exact Mie scattering [Mie09] from spherical water droplets. Except for the first bounce from the light source, and the last bounce to the viewpoint, which use one exact direction each, all intermediate bounces are via the array `bintobin`, so any phase function can be used efficiently.

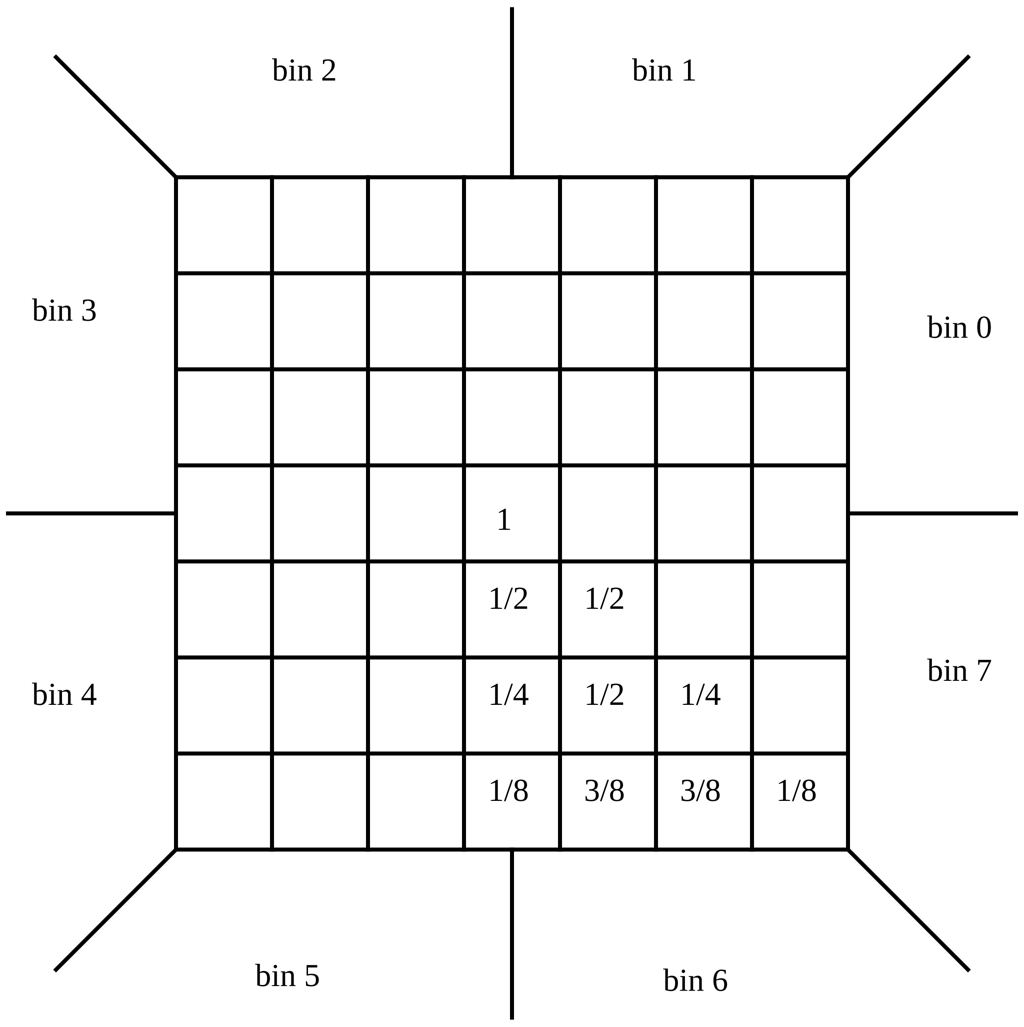

Figure 1. Direction bins and binomial weight distribution.

3. Simultaneous Shooting

Consider a unit cube of un-normalized direction vectors, with each face divided into $2m \times 2m$ equal bins, giving a total of $M = 24m^2$ direction bins. The previous section assumed that the flux `through[k][l]` is uniformly distributed in the direction bin B_l, so that the intensity is constant for any direction inside the bin. From now on, I assume that the flux within a single bin is proportional to surface area on the unit direction cube. This assumption is necessary for the efficient method, described next, for propagating the flux incrementally from a cell to its close neighbors. It introduces some error in the light distribution, but the error decreases with decreasing bin size.

For simplicity, I will first discuss the method for the 2-D, $m = 1$ case shown in figure 1. There are $M = 8m = 8$ direction bins. I will first describe a simple scheme for propagating the flux which gives a binomial distribution, and then show how to modify it to give a uniform distribution. Consider the bin between 270° and 315° (bin 6 counting from 0), and suppose the pixel in the center of the square has a unit flux leaving within this bin. Approximately half of this flux enters the pixel below, and half enters the one diagonally below and to the right, so these two pixels are marked in figure 1 with the weights 1/2. If each of these pixels distributes its flux in the same (1/2, 1/2) scheme to the row below, the flux in that row would be 1/4, 1/2, 1/4, as shown. In the third row below the shooting pixel, the pattern is 1/8, 3/8, 3/8, 1/8. In general, at pixel (i, n), the ith pixel in the nth row below the shooting pixel at (0, 0), the weight is $\frac{1}{2^n}\binom{n}{i}$, giving a binomial distribution. The binomial coefficient $\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$ counts the number of n-step paths from (0, 0) to (i, n), obtained by taking any i of the steps in a diagonal direction, and the rest directly downwards.

This binomial (1/2, 1/2) scheme distributes the flux across the bin, but not in the uniform way desired. The desired distribution is shown in figure 2, with the pixels in row 3 marked with the weights 1/2, 1, 1, and 1/2. The outer pixels are counted half in this bin and half in adjacent bins.

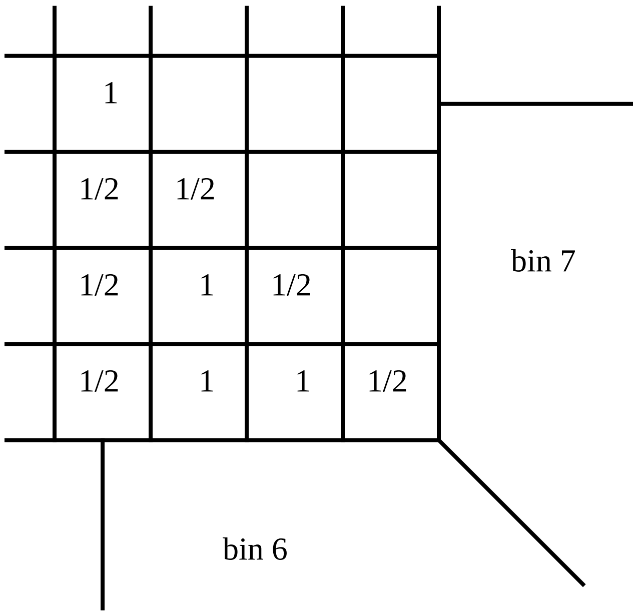

Figure 2. Bin 6 from figure 1, with desired weight distribution.

The sum of these weights is 3, so they must be normalized by dividing by 3 to give the portion of the bin's flux reaching each pixel. To propagate this weight pattern to the next row below, first add 1/2 to each of the outer two pixels, to get a pattern of all 1's. Then add each of these 1's, half to the pixel below, and half to the pixel below and to the right. The result is the desired pattern 1/2, 1, 1, 1, 1/2 in the fourth row.

This pattern of weights, when normalized, shows the proportion of the shot flux reaching a receiving pixel in the absence of intervening opacity. To account for opacity, each value is multiplied by the transparency at the current pixel, before propagating by the (1/2, 1/2) scheme, so that the opacity is accumulated along the propagation path. The added adjustment to the left most pixel is similarly attenuated by the opacities in the column above it, and the adjustment to the right most pixel, by the opacities along a 45° diagonal. The iterative process actually starts with the outgoing flux to be shot in a direction bin, instead of the unit flux discussed above. In row *s*, it builds a pattern of *s*+1 appropriately weighted and attenuated values, which are divided by *s* and added into the `receive` array at that row.

The arithmetic involved in this iteration is independent of the horizontal displacement between the shooting and the receiving pixel, so it can be done simultaneously for each pixel in a horizontal row, as indicated by the code fragment in figure 3.

```
/* Initialize with unshot radiosity from row i0. */
for (j = 0; j < columns; ++j) {
   work[j] = unshot[i0][j];
   corner[0][j] = unshot[i0][j];
   corner[1][j] = unshot[i0][j];}
for (i = i0 + 1; i < rows; ++i) {
/* Propagate radiosity to row i. */
   for (j = 0; j < columns; ++j) {
      s = i - i0;
      tempwork[j] = .5 * work[j];
      if (j > 0) tempwork[j] += .5*work[j-1];
      receive[i][j] = tempwork[i][j]/s;
/* Adjust tempwork using corner arrays. */
      tempwork[j] += .5 * corner[0][j];
      if (j > 0) tempwork[j] += .5 * corner[1][j-1];}
/* Update work and corners, to account for transparency. */
   for (j = 0; j < columns; ++j) {
      transparency = exp( - raylength * kt[i][j] );
      work[i][j] = tempwork[i][j] * transparency;
      corner[0][j] *= transparency;
      if (j > 0) corner[1][j] = corner[1][j-1] * transparency;
      else corner[1][j] = 0.; } }
```

Figure 3. Code fragment for 2-D flux propagation.

The actual implementation contains subscripts indicating the bin direction and starting row, not shown in the fragment. These let multiple directions and starting rows be propagated together, permitting multiple bounces per pass, as discussed below.

The array `work[]` stores the flux propagating in the direction bin 6 in figure 2, and is initialized with the unshot flux from row *i*0, `unshot[i0][]`. The array `corner[0][]`, named after its 3-D use, stores the flux propagating directly downward, used to adjust the left-most pixel in figure 2, and the array `corner[1][]`stores the flux propagating diagonally, to adjust the right-most pixel. The constant `raylength` is the average length of a ray/pixel intersection segment, and depends on the bin index. For a square collection of pixels, with `rows` = `columns` = *n*, this code fragment computes the O(n^3) interactions of the shooting row with the pixels below it in time O(n^2), instead of the O(n^4) time which would be required to accumulate the opacity for each interaction along a straight path of length O(n). However this n^2 savings factor comes at a cost in accuracy. The attenuation is not accumulated only along the straight path between a shooting and receiving pixel, but instead along the many possible propagation paths of downward and diagonal steps connecting them. Several of these paths are shown in figure 4, filling out a parallelogram.

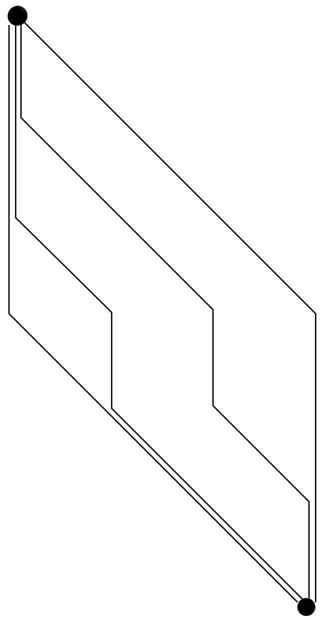

Figure 4. Paths connecting two pixels.

The opacities at all the pixels in this parallelogram will influence the occlusion of the flux shot from pixel (0, 0) and received at pixel (i, n). The opacity at pixel (j, k) contributes according to the number of propagation paths passing through it. Paths belonging to the simple (1/2, 1/2) scheme are weighted by $1/2^n$, while those which first go l steps along one of the two `corner` arrays, being divided by 2 only on the last step, are weighted by $1/2^{n-l+1}$. Thus, neglecting the nonlinearity of the exponential function, the opacity contribution from pixel (j, k), when $0 < j < k$, is

$$\frac{1}{2^n}\binom{n-k}{i-j}\left[\binom{k}{j} + \sum_{l=1}^{k-j} 2^{l-1}\binom{k-l}{j} + \sum_{l=1}^{j} 2^{l-1}\binom{k-l}{j-l}\right].$$

The binomial coefficient outside the square brackets represents the number of paths from (j, k) to (i, n). The three binomial coefficients in the square brackets represent the number of paths from (0, 0) to (j, k) using, respectively, the (1/2, 1/2) scheme alone, l steps of vertical `corner` propagation, or l steps of diagonal `corner` propagation.

This opacity contribution is show as a function of the location (j, k), with k increasing downwards, for $n = 25$, and $i = 5$, 9, 13, and 17, in figures 5*a* through 5*d*, respectively. Black denotes the greatest contributions, and the palest grey is used for any non-zero contribution. The $i = 13$ case in figure 5c shows mainly the effect of the binomial (1/2, 1/2) scheme alone. The weight is concentrated near the straight path, but spread somewhat, blurring the shadows. For the other i

shown, the two summations giving the contributions from the `corner` arrays bias the contribution toward one of the `corner` directions, giving extra weight to shadowing objects in these directions. I believe these continuously varying shadow errors are less serious than the discontinuous illumination errors due to the ray effect.

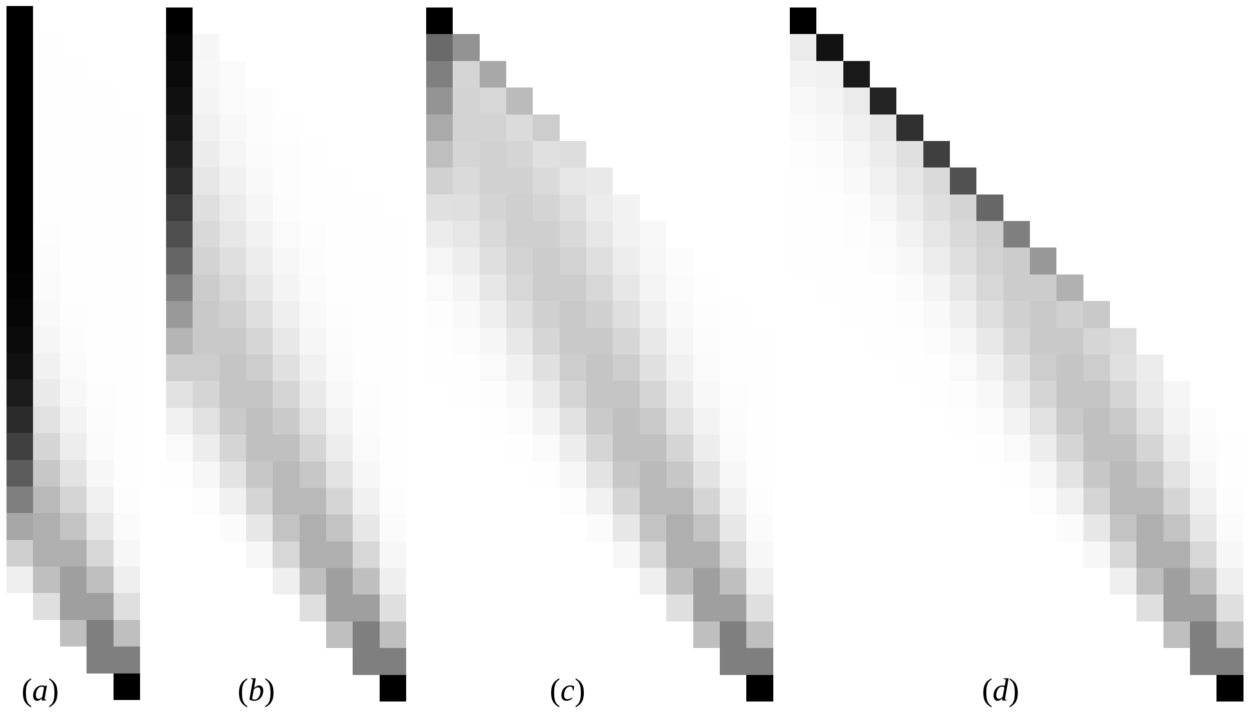

Figure 5. The number of paths passing through each cell.

4. The 3-D case

In the 3-D case, the flux in a direction bin shot from a central voxel spreads out across the faces of a cubical shell. Figure 6 shows the weights for one of the 24 direction bins in the $m = 1$ case, marked on a surface layer of a $7 \times 7 \times 7$ cubical shell. Voxels shared between two adjacent bins are marked with weight 1/2, and those shared between four adjacent bins are marked with weight 1/4. Only voxels at the eight cube corners are shared between three adjacent bins, and are marked with weight 1/3. These corners require a separate correction.

First consider the case where all four corners have weight 1/4. The analogy to the 2-D case should be clear. The pattern of weights of value 1, 1/2, and 1/4 can easily be constructed from the smaller pattern of all 1's in the next shell inwards. Simply divide each value in the smaller pattern by 4, and add it to the appropriate four direct or diagonal neighbors. The procedure to reconstruct the all 1's pattern is a little more complicated, since four whole edges of weight 1/2 must be added on. The needed `edge` arrays can be maintained by the 2-D procedure described in the preceding section. These 2-D iterations require four `corner` arrays, along the four corners of the direction bins. The four `corner` arrays are also used to adjust the corner values of the pattern of weights to exactly 1, since the addition of the `edge` arrays leaves them off by 1/4. The weighted attenuated `work` values s layers beyond the shooting plane are divided by s^2 and added into the `receive` array at that layer. Finally, $1/(12s^2)$ times the `corner` array, if any, corresponding to a cube main diagonal direction is added to `receive` to make the final weight 1/3.

			1/4	1/2	1/2	1/4
			1/2	1	1	1/2
			1/2	1	1	1/2
			1/4	1/2	1/2	1/3

Figure 6. Weights for a layer in a 3D bin.

The temporary `work`, `edge`, and `corner` arrays are initialized with the n^2 `unshot` direction bin flux values in a shooting plane, and propagate their flux to O(n^3) receiving elements, using O(n^3) time to produce O(n^5) interactions. Thus the total cost for propagating a single bin direction for the n shooting planes is O(n^4). This last factor of n can be reduced to O(log n) by maintaining these temporary arrays from all the shooting layers, as the receiving layer progresses through the volume, and recursively consolidating them when the resulting error is small.

The only computational difference between the treatment of the various shooting layers is the inverse square factor $1/s^2$. Suppose we take the flux in the `work`, `edge`, and `corner` arrays for a shooting layer at separation s from the current receiving layer, and at each entry, put half the flux into the corresponding entry in the array for the shooting layer at separation s - a, and half into the corresponding entry for separation $s + a$. These two layers each have their own inverse square factor, so the effective inverse square factor will become

$$\frac{1}{2\,(s-a)^2} + \frac{1}{2\,(s+a)^2} = \frac{s^2+a^2}{(s^2-a)^2} = \frac{1}{s^2}\left(\frac{1+a^2/s^2}{(1-a^2/s^2)^2}\right). \tag{6}$$

We start with $a = 1$, and redistribute layers with odd separations into layers with even separations.

Given an error tolerance e, we can find a separation s_0 beyond which this consolidation results in a “form factor” error of less than e. This flux consolidation can be continued recursively. At the ith level of recursion, we redistribute the flux in layers of separation $s = t_0 + 2^{i+1} + 2^i k$, for $k = 1, 3, 5, \ldots$, onto layers of separations $s - a$ and $s + a$, with $a = 2^i$. Using equation (6), one can show that for sufficiently large t_0, independent of the volume array side n, the total error introduced is less than e. The number of layers remaining after this consolidation is O(log n).

It is actually possible to reduce the O(log n) factor to O(1) by consolidating the voxels within a layer, as well as between adjacent layers, in the manner of [Hanr91] and [Bhat93]. The additional errors would not be large, because the occlusion effects at large distances become fuzzy, as shown in figure 5. I have not coded this enhancement, because the practical differences between O(log n) and O(1) are small.

5. The $m > 1$ case

For $m > 1$, the propagation is more complicated. The `work`, `edge`, and `corner` arrays are maintained only for separations s divisible by m. For cells i and k with separations less than m, more accurate galerkin type geometric form factors $Geom_{ijk}$ are precomputed using Monte-Carlo integration, in place of the approximation in equation (3). These are used to propagate the flux from the `work`, `edge`, and `corner` arrays at separations nm to get the `receive` flux at cells at separations $nm + 1$ up to $nm + m - 1$, and to account for the effect of the opacities in these cells on these arrays. The `work`, `edge`, and `corner` are then updated as discussed above, to propagate the flux to cells at separation $(n + 1)m$. I have implemented the $m = 2$ case, with 96 direction bins in 3-D, and used it to produce all the results in this paper.

6. Scattering of received flux

Once the flux in a direction bin is received in a cell, it must be added to the tally in `through`, for use in a final extra bounce towards the viewpoint during rendering. It must also be scattered into the `unshot` flux in each of the M direction bins at the receiving cell, using one row of the $M \times M$ scattering matrix `bintobin`. This costs time M, so the total cost per direction bin is $O(n^3 \log n + Mn^3)$. The log n is missing from the second term, since the flux from all shooting layers is maintained during one pass through the volume, and consolidated into `receive`. A pass through all M direction bins costs time $O(Mn^3 \log n + M^2 n^3)$.

Note that in equation (5), the product of all the factors after the albedo a is less than 1, so after k bounces, the flux is decreased by a factor of at least a^k. For a fixed albedo a less than 1, the error can thus be made smaller than a set tolerance after a number of passes that depends only on a, and not on n, that is, in O(1) passes. For scattering from water droplets in clouds, a is very close to 1, which would theoretically make the O(1) iteration count very large. In practice, the flux leaks out at the edges of the cloud, so there is reasonable convergence even when $a = 1$.

The finite element implementations of Rushmeier set the form factor between a volume element and itself to zero [personal communication], because her inverse square approximation to the form factor had a singularity in this case. However, to approximate dense clouds with large enough elements for practical computation, it is necessary to account for scattering within a single element. To do so, I assume exactly forward scattering, a fairly good approximation for water droplets, in order to calculate the probability of higher order scattering. The multiple scattering events are then governed by a Poisson distribution [Fell68]. Let l be the average length of the intersection of a ray in the incoming direction bin with a volume element cube of opacity k_t per unit length, and let $\lambda = k_t l$. Then the probability that the ray emerges unscattered is $e^{-\lambda}$ and the probability of emerging after b bounces is $\lambda^b e^{-\lambda}/b!$. At each cell the flux in `receive` from direction bin B_i is distributed as `unshot` and `through` flux into all bins B_j for that cell by the factors

$$\sum_{b=1}^{B} \frac{a^b \lambda^b}{b!} e^{-\lambda} \texttt{bintobin}^b\texttt{[i][j]},$$

where a is the albedo, and the powers of the `bintobin` matrix are precomputed. The number B of terms required depends on a and the range of k_t but is O(1) as a function of n. For the images in the results section, I used $B = 12$ terms.

This approximation was checked by comparison with the Monte Carlo simulation described by Hanrahan and Krueger [Hanr93], and agreed well even when the scattering was not forward. (The box on page 170 of [Hanr93] giving the Monte Carlo simulation has three errors, confirmed by the authors. The absolute value signs around $2g$ in the expression for cos j should be removed, the last row in the vector $\bar{t}$ should be - cos φ sin θ, and there is no need to adjust the weight using the distance to the boundary if d causes the particle to leave the layer.)

The multiple scattering within one cell speeds up the convergence of the iteration. Another way to speed up the convergence is to process multiple scattering events at different cells during one sweep through the volume. The $4m^2$ shooting bins in one of the six faces of the direction cube are processed together as the receiving planes sweeps along the corresponding axis direction. The energy scattered from one direction bin to another in the same cube face can then be processed for further transmission and scattering during the same sweep. When the scattering is predominately forward, the scattered flux is likely to end up in a direction in the same cube face.

In order to maintain the O(log n) temporary arrays of size n^2 for each of the O(M) directions on a cube face, O(Mn^2 log n) storage is required. This is asymptotically less than the O(Mn^3) needed for the `through` and `unshot` arrays.

7. Final gathering pass

The final rendering uses an evaluation of the integral form of the transport equation along a ray through each pixel, as a summation over the ray/element intersection segments. In this final gathering step, I displaced the volume cells so that their vertices were at the centers of the original elements, and used interpolated values of `through` to give smoother shading. To use the exact direction ω of the viewing ray, instead of just its direction bin, I computed the integrals

$$\int_{B_l} f(\omega, \omega')\, d\omega'$$

for each bin B_l once per viewing ray, or else once per volume element, depending on which are less numerous. This gives a smooth variation of the scattering with the viewing angle.

Similarly, the `unshot` flux is initialized from the attenuated light source flux array `en`, using integrals involving the exact direction to the light source. To compute `en`, many illumination rays are traced through the volume, enough to cross each volume element multiple times. The ray/element intersections are processed in the order of light propagation, to attenuate the intensity by the element opacity, and to add the flux into `en`.

Note that the light source flux in `en` is not transferred to `through`. The final gathering pass computes the single scattering contribution using this accurately attenuated direct illumination, without the shadow blurring caused by the spread out opacity weighting shown in figure 5. For this single scattering, the phase function is evaluated using the exact directions of both the viewing and the direct illumination rays.

8. Cloud model

The geometry of the cloud is determined by the density array `kt`. Kajiya and Von Herzen [Kaj84] computed this density with a meteorological simulation. Instead, for the purposes of test rendering, I used a variant of the visual cloud model of Gardner [Gard84]. Gardner rendered the surfaces of ellipsoids with a 3-D transparency texture based on a pseudo-fractal trigonometric series. I wanted an analogous 3-D density function. I took quadratic polynomials of the form

$$d - \frac{(x - x_0)^2}{a^2} - \frac{(y - y_0)^2}{b^2} - \frac{(z - z_0)^2}{c^2}$$

whose contours are ellipsoids, and used the maximum of several such ellipsoidal functions with different parameters to define the union of ellipsoidal clouds. I then added on a version of Ken Perlin’s $1/f$ noise function [Perl85], to roughen and randomize the edges. Like Gardner, when the volume function was negative, I let `kt` = 0, giving complete transparency. More sophisticated cloud turbulence models are given in [Sak93] and the references therein.

9. Results

Figure 7 shows a cloud with the sun behind it, rendered with multiple anisotropic scattering. Note that the cloud edges are brightest near the direction of the sun. Figure 8 shows the same cloud from a different direction, with the green “grass” background color added for orientation. For comparison, figure 9 shows the view in figure 8 with only single anisotropic scattering, and figure 10, the difference of figure 8 minus figure 9, indicates the contribution of the higher order scattering.

The cloud was defined on a $24 \times 24 \times 18$ voxel volume. The initial illumination pass, with approximately 1000 illumination rays per voxel, took 120 seconds on an SGI 4D/35. The albedo was .99, and the Henyey-Greenstein phase function had $g = .55$, for forward scattering. I used 96 direction bins. Each of the 15 scattering passes took 15 minutes. The final rendering at 500×384 resolution took 5 minutes per frame. Once the multiple scattering flux in `through` has been computed, frames can be rendered from any viewpoint, so the 300×225 resolution frames on the videotape took an average of two minutes each.

Figure 11 shows a side view and a top view of the cloud at sunset, using two light sources, an orange one from near the horizon representing the sun, and another bluish one representing the sky illumination. These frames took twice as long for the two passes. For the sky illumination, I used the CIE standard clear sky directional luminance function [CIE37] to initialize the `unshot` array on an extra shell of cells on the top and sides of the volume. Figure 12 shows a top view of another cloud, using an orange point source for the setting sun, and a blue point source for the sky.

10. Future work

This method should be applicable to engineering computations if black body emission is included in the flux propagation, an easy modification.

Rushmeier [Rush88] and Kajiya [Kaj84] have pointed out that after a number of scattering events, even a narrow forward phase function becomes more isotropic. This means that the later scattering passes through the volume could use a smaller number of direction bins, for greater speed, and still maintain accuracy.

Rushmeier [Rush 87, Rush88] handles surface and volume radiosity in a unified framework. I currently do not handle surface radiosity, but it should be possible to include surface elements in this method. In a common engineering application, the only surfaces are on the enclosure of the participating medium. In this straightforward case, a directional pass through the volume begins with the unshot flux leaving a shooting surface, as described above for the sky illumination, and the flux exiting the sides or left over at the end is deposited on the appropriate receiving surface.

Langer *et al*. [Lang93] have applied the discrete ordinates method to general surface geometries, using “surface nodes” with a bidirectional reflection distribution function at voxels containing surfaces. They can thus include anisotropic surface reflections, as well as isotropic volume scattering and absorption. Their flux propagation, like mine, is along a discrete cube of directions, and could be enhanced by my method to reduce the ray effects.

11. Acknowledgments

This work was performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under contract W-7405-ENG-48. I would like to thank Holly

Rushmeier, John Howell, Roger Crawfis, and Charles Grant for helpful conversations, Chris Patmore, Steven Zucker, and Neeta Bhate for preprints of [Patm93], [Lang93] and [Bhat93], respectively, and anonymous Siggraph and Eurographics reviewers for suggesting improvements and pointing out references [Blas93], [Sak93], and [Kan90].

References

[Bhat92] N. Bhate and A. Tokuta, “Photorealistic Volume Rendering of Media with Directional Scattering”, Third Eurographics Conference on Rendering (May 1992) Consolidation Express, Bristol England, pp. 227-245

[Bhat93] Neeta Bhate, “Application of Rapid Hierarchical Radiosity to Participation Media”, in “ATARV-93: Advanced Techniques in Animation, Rendering, and Visualization” (B. Özgüç and V. Akman, eds.), Bilkent University (July 1993) pp. 43 - 53

[Blas93] P. Blasi *et al.* “A Rendering Algorithm for Discrete Volume Density Objects,” Eurographics ’93 (1993) c-202

[CIE73] CIE Technical Committee 4.2: Standardization of Luminance Distribution on Clear Skies, CIE Publication 22, Commission International de l’Eclairage, Paris (1973) p. 7

[Chan50] Subrahmanyan Chandrasekhar, “Radiative Transfer”, The Clarendon Press, Oxford, (1950) (or Dover Press, New York, 1960)

[Coh88] Michael Cohen, Shenchang Eric Chen, John Wallace, and Donald Greenberg, “A Progressive Refinement Approach to Fast Radiosity Image Generation”, Computer Graphics Vol. 22 No. 4 (August 1988) pp. 75 - 84

[Fell68] William Feller, “An Introduction to Probability Theory and its Applications, Volume I, Third Edition”, John Wiley & Sons, Inc., New York (1968)

[Gard84] Geoffrey Gardner, “Simulation of Natural Scenes using Textured Quadric Surfaces”, Computer Graphics Vol. 18 No. 3 (July 1984) pp. 11 - 20

[Gort93] Steven Gortler, Michael Cohen, and Philipp Slusallek, “Radiosity and Relaxation Methods: Progressive Radiosity is Southwell Relaxation”, Technical Report, Computer Science Dept., Princeton University (shortened version to appear in IEEE CG&A, 1994)

[Hanr91] Pat Hanrahan, David Salzman, and Larry Aupperle, “A Rapid Hierarchical Radiosity Algorithm”, Computer Graphics Vol. 25 No. 4 (July 1991) pp. 197 - 206

[Hanr93] Pat Hanrahan and Wolfgang Krueger, “Reflection from Layered Surfaces due to Subsurface Scattering”, Computer Graphics Proceedings, Annual Conference Series (1993) pp. 165 - 174

[Heny40] G. L. Henyey and J. L. Greenstein, “Diffuse Radiation in the Galaxy”, Astrophysical Journal Vol. 88 (1940) pp. 70 - 73

[Imm86] David Immel, Michael Cohen, and Donald Greenberg, “A Radiosity Method for Non-Diffuse Environments”, Computer Graphics Vol. 20 No. 4 (1986) pp. 133 - 142

[Inak89] M. Inakage, “An Illumination Model for Atmospheric Environments”, New Advances in Computer Graphics: Proceedings of C. G. International ’89 (R. A. Earnshsaw and B. Wyvill, eds.) Springer Verlag, Tokyo (1989) pp. 533 - 548

[Kaj84] James Kajiya and Brian Von Herzen, “Ray Tracing Volume Densities” Computer Graphics Vol. 18 No. 3 (July 1984) pp. 165 - 174

[Kan90] Kazufumi Kaneda, Takashi Okamoto, Eihachiro Nakamae and Tomoyuki Nishita, “Highly Realistic Visual Simulation of Outdoor Scenes under Various Atmospheric Conditions”, CG International ’90, (T. S. Chua and T. L. Kunii, eds.) Springer-Verlag, Tokyo (1990) pp. 117 - 131

[Kauf87] Arie Kaufman, “Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes”, Computer Graphics Vol. 21 No. 4 (July 1987) pp. 171 - 179

[Lath68] K. D. Lathrop, “Ray Effects in Discrete Ordinates Equations” Nuclear Science and Engineering Vol. 32 (1968) pp. 357 - 368

[Mie09] Gustav Mie, “Optics of Turbid Media”, Ann. Physik Vol. 25 No. 3 (1908) pp. 377 - 445

[Nish87] Tomoyuki Nishita, Yasuhiro Miyakawa, and Eihachiro Nakamae, “A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources” Computer Graphics Vol. 21 No. 4 (July 1987) pp. 303 - 310

[Lang93] Michael Langer, Pierre Breton, and Steven Zucker, “Parallel Radiosity without Form Factors”, report TR-CIM-93-22, McGill Research Center for Intelligent Machines, McGill University, Montreal, Quebec, Canada (December 1993)

[Patm93] Chris Patmore, “Simulated Multiple Scattering for Cloud Rendering” in “Graphics, Design, and Visualization: Proceedings of the International Conference on Computer Graphics -ICCG93”, S. P. Mudur and S. N. Pattaniak, eds., Elsevier Science Publishers (1993) pp. 29-40

[Perl85] Ken Perlin, “An Image Synthesizer”, Computer Graphics Vol. 19 No. 3 (July 1985) pp. 287 - 296

[Rush87] Holly Rushmeier and Kenneth Torrance, “The Zonal Method for Calculating Light Intensities in the Presence of Participating Media”, Computer Graphics Vol. 21 No. 4 (July 1987) pp. 293 - 302

[Rush88] Holly Rushmeier, “Realistic Image Synthesis for Scenes with Radiatively Participating Media”, Ph.D. Thesis, Cornell University (May 1988)

[Sak93] Georgios Sakas, “Modeling and animating turbulent gaseous phenomena using spectral synthesis”, The Visual Computer Vol. 9 No. 4 (January 1993) pp. 200 - 212

[Sie92] Robert Siegel and John Howell, “Thermal Radiation Heat Transfer, Third Edition”, Hemisphere Publishing Corp., Washington (1992)

[Tan89] Zhiqiang Tan, “Radiative Heat Transfer in Multidimensional Emitting, Absorbing, and Scattering Media –- Mathematical Formulation and Numerical Method”, Journal of Heat Transfer Vol. 111 (February 1989) pp. 141 - 147

[Voss83] Richard Voss, “Fourier synthesis of gaussian fractals: 1/*f* noises, landscapes, and flakes”, Tutorial on State of the Art Image Synthesis, ACM Siggraph Course Notes (1983)

[Will92] Peter Williams and Nelson Max, “A Volume Density Optical Model”, Proceedings, 1992 Workshop on Volume Visualization, ACM Press, New York (1992) pp. 61 - 68

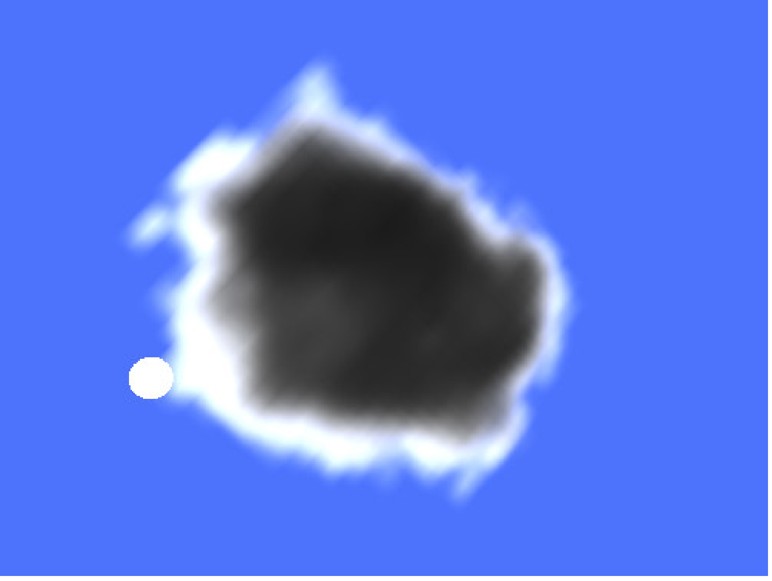

Figure 7

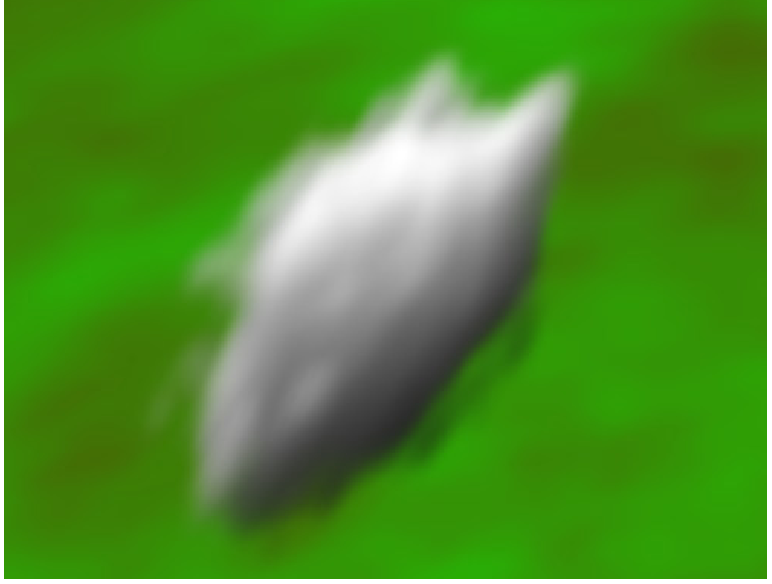

Figure 8

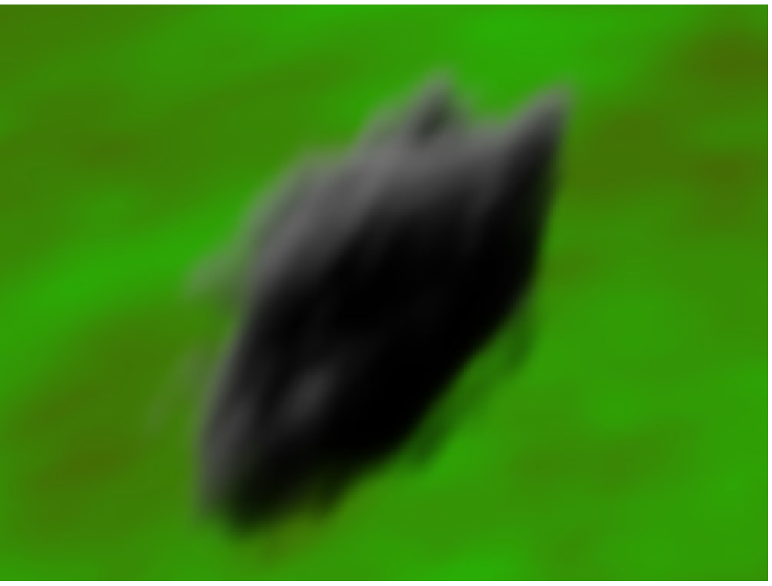

Figure 9

Figure 10

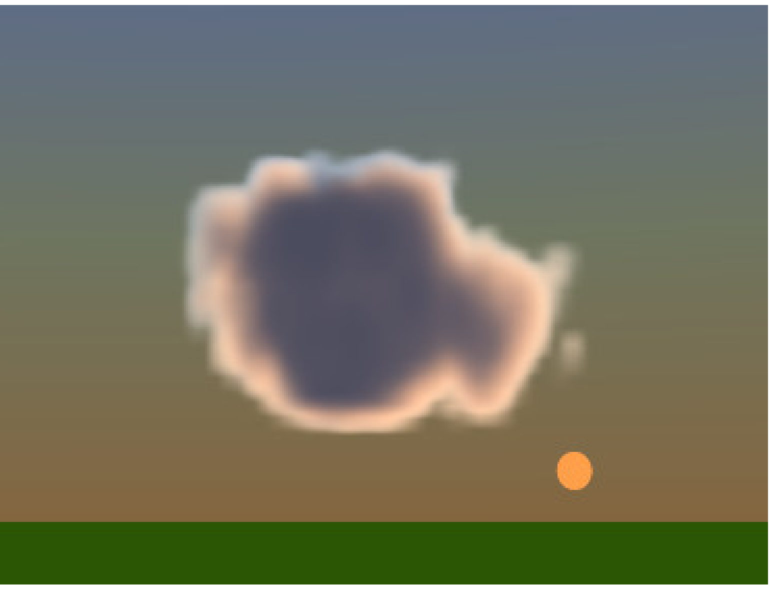

Figure 11

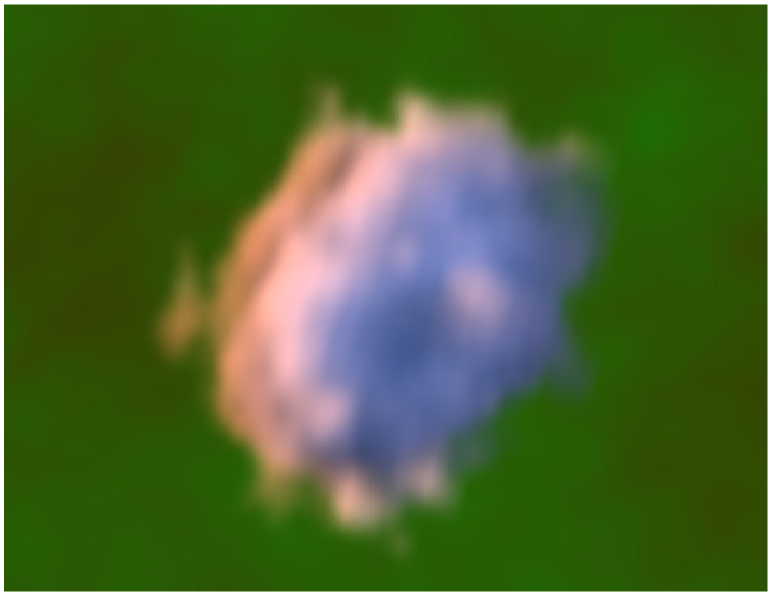

Figure 12